simul_run
User's manual

Miguel Hernandez University [1]

April 19, 2011

Abstract

In this manual the installation and usage of the *simul_run* program is described, which is included within the *simul_xxx* set of programs. It has been developed by P. Pablo Garrido Abenza (pgarrido@umh.es), as a member of the Arquitectura y Tecnología de COMputadores Group (GATCOM) at the Miguel Hernandez University.

Contents

Chapter 1

Introduction

For executing simulations with OPNET Modeler [1] it can be used:

- The OPNET Modeler Graphical User Interface (GUI).
- From the command-line with the `op_mksim` or `op_runsim` tools.

Although using interactively the GUI is easier than the other options, it has the disadvantage of the overhead of memory used by the graphical application itself, whereas from command-line, more memory is available.

On the other hand, in order to execute a sequence of simulations, we have two possibilities:

- To use the 'Simulation Sequence Editor' within the OPNET Modeler GUI, which is shown when a new file is created (File > New File) selecting 'Simulation Sequence' as the type of file. It also appears instead of the 'Configure/Run DES' dialog box when the 'des.configuration_mode' preference is set to 'advanced'.
- To write shell scripts (Unix) or batch files (MS-DOS), which run outside the OPNET Modeler.

As it has been said before, to use the GUI requires left less memory available to the simulations. In addition, the use of scripts is more flexible, as they allow to run other commands before launching the simulations, for example, to write input files or arguments on the fly. However, in order to write scripts it is necessary to know the specific script language for the platform used (MS-DOS/Unix) and command-line interpreter: MS-DOS, Bourne shell (sh o bash), C shell (csh), Korn shell (ksh), ... As all of them are very different between them, it is necessary to rewrite all the developed scripts in order to use other platform. Moreover, these languages have the drawbacks that they are very limited (some more than others), complicated and difficult to remember or understand syntax, and the scripts are not easy to maintain.

Some sample scripts can be seen in the 'Simulation Scripts' section (or 'Simulation Sequences > Scripted Sequences') in the on-line OPNET Modeler documentation. A typical situation is to run a sequence of simulations varying the seed value

in order to achieve a high degree of confidence in the output mean statistic values. For example, the following script presented in that section can be rewrited to a DOS-batch file:

```
#!/bin/csh
# seed_seq.csh: script to run simulations with a varying seed
# ----------------------------------------------------------
@ seedval = 1
while ($seedval < 101)
echo 'Executing Simulation with Seed: '$seedval
op_runsim -net_name mymodel -seed $seedval
@ seedval = ($seedval + 1)
end
# ----------------------------------------------------------
```

```
@rem ------------------------------------------------------------------
@rem seed_seq.bat: batch file to run simulations with a varying seed
@rem ------------------------------------------------------------------
@rem SET SEEDS=1 2 3 ... 100
@rem for %%seed in (%SEEDS%) do op_runsim -net_name mymodel -seed %%seed
@rem SET SEEDS=
for /l %%seed in (1 1 100) do op_runsim -net_name mymodel -seed %%seed
rem -------------------------------------------------------------------
```

The aim of the *simul_run* program is to define and run a sequence of simulations in an easier way and in the same manner in all the platforms. Only is necessary to be familiarized with all the options required by *simul_run*, and as it will be shown below, a great part of those options are similar as used for 'op_runsim', and they will be pass through to it. With the execution of this program with the appropriate arguments, a complete simulations set, varying the model scenario, values for promoted attributes (via different environment files .ef), and with different random number seeds. Perhaps the only manual step would be to prepare those .ef files, but they can be copied from a previous one created during a short simulation within the OPNET Modele GUI, then changing the varying values in each case. Another possibility would be to create our own program for the generation of such .ef files.

In addition, this program (as the other ones within the *simul_xxx* packaqe) can be executed in a Condor cluster [2], by only specifing the `-cluster` option. In this way, the user can execute so many simultaneous simulations as number of processors available (limited also by the number of available OPNET Modeler licenses), without needing any knowledge about launching jobs in Condor, neither writing any script file needed by Condor (.job or .dag files).

The set of programs *simul_xxx*, contains the following command-line tools:

- *simul_geoloc*: used for generating network scenarios automatically, computing the partition grade according to the transmission range.
- *simul_ah2r*: used for converting the binary animation file (.ah) to a text format (.as).
- *simul_run*: used for program sets of simulations with OPNET Modeler without using the Graphical User Interface.

- *simul_data*: used for extracting the data obtained with the simulations. This program calls to *ov2txt* for each of the simulations run.
- *ov2txt*: invoked from the *simul_data* for extracting data from the binary vectorial files (.ov). It can also be invoked for the user for extracting data for a particular simulation.
- *simul_graphs*: used for generating graphs from the extracted data (R or gnuplot).
- *simul_reports*: used for generating a report containing the previously generated data and graph.

On the contrary to the conventional MS-DOS/Unix script languages, these programs have been developed using the C programming language, so it has been easy to migrate them to other platforms, namely, Windows and Unix-like systems like Linux or Mac OS X[1].

[1]Currently OPNET Modeler is only available for Windows y Linux, although perhaps it could be executed on Mac OS X by using X.

Chapter 2

Installation

For the instalation of the *simul_run* program, you only have to copy the binary to any directory and set some environment variables (`PATH` and `LD_LIBRARY_PATH`). In the following sections, how to set these environment variables is explained.

2.1 Requeriments

The program *simul_run* requires that OPNET Modeler be installed, as it makes use of the API External Model Access (EMA). Moreover, the OPNET Modeler version installed should match with the one used for compiling this program, which is specified in the name of the executable (e.g. ov2txt-...-16.0.A.bin).

2.2 Windows

Let us suppose that the directory choosen for copying the *simul_run* program is:

```
C:\Program files\utils
```

It is recommended to modify the `PATH` environment path. This can be do as explained in the following:

- Windows 98 or previous: edit the C:\AUTOEXEC.BAT file, adding the following line at the end. It would be possible that the path needs to be enclosed within quotes if it is more than 8 characters length or it contains blanck spaces:

  ```
  SET PATH=%PATH%;C:$\Program files\utils
  ```

- Windows NT, 2000, XP, ...: within the Control Panel, choose System > Advanced > press the button [Environment variables]. Then the path could be added to the `PATH` environment variable, separated by a semicolon ';'. Alternatively, the Autoexec.NT file can be modified in a similar way as explained for the Autoexec.bat file.

2.3 Linux

Let us suppose that the directory choosen for copying the *simul_run* program is:

```
$HOME/bin
```

Under Linux it is necessary to set the `PATH` and `LD_LIBRARY_PATH` environment variables. In the later we should to add the path for the OPNET libraries, as it is necessary in order to run programs compiled with the API EMA (built with op_mkema). This path depends on the home directory for OPNET and version. The way to do that depends on the shell or command-line interpreter used (bash, ksh, ...):

- Linux (shells: bash, ksh, zsh, sh): $HOME/.bash_profile or $HOME/.profile:

```
PATH=.:$PATH:$HOME/bin
LD_LIBRARY_PATH=/usr/opnet/16.0.A/sys/pc_intel_linux/lib
export PATH
export LD_LIBRARY_PATH
```

- Linux (shells: csh, tcsh): $HOME/.login:

```
setenv PATH .:$PATH:$HOME/bin
setenv LD_LIBRARY_PATH /usr/opnet/16.0.A/sys/pc_intel_linux/lib
```

Some Linux distributions have problems with the `LD_LIBRARY_PATH`, and user cannot set the value. This is why it is necessary to execute that each time that either you are going to execute the program or you open the console.

In order to simplify the process, you can write a little script called ov2txt.sh similar to the following, and put it together to the binary. In order to pass the arguments from the script to the executable, you can use `"$@"` (with the quotes) instead of `$*` too.

```
#!/bin/sh
export LD_LIBRARY_PATH=/usr/opnet/16.0.A/sys/pc_intel_linux/lib
exec simul_run-en-16.0.A-x86.bin $*
```

Chapter 3

Usage of the program

In this chapter, the usage of the *simul_run* program is explained. It is supposed that we have written the *simul_run*.sh script explained in the previous chapter.

3.1 Syntax

```
simul_run -?
simul_run -m <input-file> [-i <input-path>] [-o <output-path>] [-w <working-path>]
          [-v] [-t <title>] [-duration <secs>] [-probe_start_time <secs>]
          [-verbose_sim] [-anim_hist] [-c]
          [-m32/-m64] [-distributed <np>]
          [-ef <file-des.ef> <num-file-params>]
          [-modelsuffixes <list>] [-seeds <list>]
          [-first <n>] [-last <n>] [-dont_run]
          [-cluster <np> <met> [<notif> <email>]]
```

3.2 Description

The *simul_run* program executes a OPNET Modeler simulations sequence, either sequentially, or in parallel (cluster Condor), or distributed (still under development). The program allows to specify a set of scenarios, to use different input parameters files (input files .ef), and to set a list of seeds. All of that define the simulations sequence to run in every distinct combination.

The program does not run interactively. Instead, the program begins to extract data immediately upon invocation and goes on until the process finishes, without interruption. Because of that, it is necessary to specify some options when invoking the program. These are the options to *simul_run*:

```
Overall arguments (help, paths, ...):
-------------------------------------
    -?: shows this help.

    -v: verbose mode.

    -i <dir>: input directory for the input files (.nt.m/.pb.m/.ef).
```

```
       If this option is not used, the input files will be searched in
       the paths specified in the 'mod_dirs' OPNET preference, in the
       same order. The use of the -i option has the advantage that the
       files processed will be the desired one, as it could exists another
       files with the same name but in different path previously defined
       in the 'mod_dirs' preference.

    -o <dir>: output directory.
       Is the destination directory for output files (ov./.os/.ef/.txt).
       This directory can be relative to the current directory (e.g. ./result)
       or absolute (e.g. /home/user/op_models/result).
       If this option is not used, the current path or the working directory
       (specified with the -w option) will be considered.
       The options -o and -w are equivalents (i.e. both produce the same
       results). The difference between them is that -w change temporarily
       the directory, and with -o, the path is used when opening the files.

    -w <dir>: change the working directory.
       Is the destination directory for output files (ov./.os/.ef/.txt).
       This directory can be relative to the current directory (e.g. ./result)
       or absolute (e.g. /home/user/op_models/result).
       By default, the working directory is the current directory from
       the program simul_run has been executed.
       It is equivalent to specify the output directory (-o option).
       The -w option is necessary when executing simul_run from a cluster.

    -t "title": title or brief description for the simulation set to run.
       It is only used for displaying information onto the screen or log files.

Arguments to pass through to 'op_runsim':
-----------------------------------------
   -duration <secs>: amount of time simulated in seconds.

   -probe_start_time <secs>: simulation time before which no probing
       takes place in seconds (se envia a op_runsim tal cual).

   -verbose_sim: the simulation prints update messages to the
       standard output during the execution.
       NOTE: contrarily to the corresponding 'op_runsim' option,
       the value true/false should not be specified, that is:
       > if it is specified, '-verbose_sim true' will be sent to 'op_runsim'
       > otherwise, '-verbose_sim false' will be sent to 'op_runsim'.

   -anim_hist: enable the animation within the simulations and be captured
       in animation history files (.ah). WARNING: these files could be huge.
       NOTE: contrarily to the corresponding 'op_runsim' option,
       the value true/false should not be specified, that is:
       > if it is specified, '-anim_hist true' will be sent to 'op_runsim'
       > otherwise, '-anim_hist false' will be sent to 'op_runsim'.

   -c: all object files referenced by the network model will be recompiled
       and the repository rebuilt prior to running the simulations (RECOMENDED).
       NOTE: recompilation only is performed before running the first
       simulation; in case of use a cluster for running parallel simulations,
       the rest of the simulations within the simulation set will wait time
       enough before start to running.

   -m32: 32-bit DES kernel will be used.
```

```
       Simulations could use up to 4GB RAM in Solaris, and 2 GB RAM in Windows
       (3 GB if the 3GB has been configured).
       NOTE: this preference is mutually exclusive with -m64; only one of these
       preferences should be used, otherwise the last specified will be used.

   -m64: 64-bit DES kernel will be used.
       Both the processor and operating system should support 64-bit.
       NOTE: this preference is mutually exclusive with -m32; only one of these
       preferences should be used, otherwise the last specified will be used.

   -distributed <np>: ejecucion distribuida de cada simulacion en varios
       procesadores. Se requiere que el sistema tenga multiple procesador.

Arguments for the simulation set:
---------------------------------
   -modelsuffixes <list>: numeric suffixes for using different scenario models
        with the same name but different suffix at the end.
        The values specified in <list> can be an enumeration or a range:
            -modelsuffixes 1,2,3,4,5,6,7,8,9,10
            -modelsuffixes 1..10

   -ef <num-file-params>]: number of input environment files (.ef)
        with values for the promoted attributes ([1..200])

        NOTE: beside these .ef files, one more .ef file will be used
        with the DES parameters (seed, etc.), e.g. mymodel_des.ef

   -seeds <list>: list of seeds to be used for generating random numbers.
        The values specified in <list> can be an enumeration or a range:
            -seeds 50,75,100,125,150
            -seeds 50..55
        In order to use this option it is necessary to comment the line
        'seed=' (with a #) in the DES .ef file (e.g. mymodel_des.ef).

   -first <n>: index for the first simulation to run.
        By default, since the first simulation of the sequence.
   -last  <n>: index for the last  simulation to run.
        By default, since the last  simulation of the sequence.

   -dont_run: do not run any simulation, just show information about the
        simulations to be run with the current specified arguments.

   NOTE: the arguments -modelsuffixes, -ef, y -seeds, define a simulation set
       with all possible combinations of the values specified.
       For example, let us suppose the following command:
          simul_run -m mymodel  -ef 2  -modelsuffixes 1..10  -seeds 50,75,100
       That command means that the following will be used:
          - 10 different scenarios: mymodel-01.nt.m .. mymodel-10.nt.m
          -  2 environment files:   mymodel_params-01.ef .. mymodel_params-02.ef
          -  3 seeds:               50,75,100
        So, 60 simulations will be run overall (10x2x3):
          #001 mymodel-01.nt.m  mymodel_params-01.ef  seed=50
          #002 mymodel-01.nt.m  mymodel_params-01.ef  seed=75
          #003 mymodel-01.nt.m  mymodel_params-01.ef  seed=100
          #004 mymodel-01.nt.m  mymodel_params-02.ef  seed=50
          #005 mymodel-01.nt.m  mymodel_params-02.ef  seed=75
          #006 mymodel-01.nt.m  mymodel_params-02.ef  seed=100
          #007 mymodel-02.nt.m  mymodel_params-01.ef  seed=50
```

```
          #008 mymodel-02.nt.m  mymodel_params-01.ef  seed=75
          #009 mymodel-02.nt.m  mymodel_params-01.ef  seed=100
           ...
          #060 mymodel-10.nt.m  mymodel_params-02.ef  seed=100

Arguments for run simulations in a cluster (Condor):
---------------------------------------------------
   -cluster <np> <met> [<notif> <email>]: run the specified simulation set
        in parallel (cluster Condor). If 'simul_run' is executed in a cluster
        and this option is not specified, the simulations (jobs) will be
        executed sequentially in the same processor.
        The arguments for this option are:
           <np>     = number of processors to be used, that is, maximun number
                      of simultaneous jobs, one per simulation (np>=0).
                      Since OPNET Modeler users must own a number of valid
                      licenses, the <np> value should be less or equal than the
                      available licenses; otherwise, many simulations will fail.
           <met>    = method used to launch the necessary jobs in Condor:
                      1-independent jobs; 2-DAG; 3-Parametrized DAG.
                      In case the number of simulations (jobs) be greater than
                      <np>, method 1 can not be used, as it could mean that.
                      there is not enough OPNET licenses for them.
                      If that case is detected, method 2 will be used.
           <notif>  = events to notify to the indicated <email>:
                      0-Never; 1-Completed; 2-Errors; 3-Always
           <e-mail> = e-mail address to send the notifications.
```

3.3 Files

The *simul_run* program will check if all the necessary input files (.nt.m/.pb.m/.ef/.trj) exist before running any simulations (invoking the 'op_runsim' tool) according to the specified options. On the other hand, the program will write several output files; the most of them are really written by 'op_runsim' (.ov/.os/.ef/.log/.ah), and one more file generated by *simul_run*(.txt) with information about the corresponding simulation to be run. Optionally, if a cluster Condor is used (`-cluster` option), the program *simul_run* will write the necessary files to launch jobs for running the simulations in parallel (.sh/.job/.dag), and the execution of those jobs will write three more output files with messages and errors (.out/.err/.log). The later will be written in a subdirectory below the current directory (or working directory).

3.3.1 Input files

1. Scenario model (.nt.m): The <input-file> is the name of an OPNET model file (option -m). The file name should be writen without the extension '.ov' at the end. If it contains blank spaces, it should be specified within quotation marks. If the scenario model specified is (e.g.) 'mymodel', it is possible that several files will be used if the -modelsuffixes is used: 'mymodel-01.nt.m', 'mymodel-02.nt.m', ... The absolute path should be included in 'mod_dirs' preference within OPNET. The file will be searched by following the same order as in 'mod_dirs'. It is possible to specify a path for the specific file to consider (-i),

which sets (temporarily) the 'mod_dirs' preference to that path.

2. Selected statistics (.pb.m): If the scenario model specified is (e.g.) 'mymodel', the file 'mymodel.pb.m' will be searched, using the OPNET 'mod_dirs' preference too.

3. Simulation DES parameters (.ef): It will be a text file named as the model followed by the suffix '_des'. For example: 'mymodel_des.ef'. It contains the text got from the DES dialog within OPNET Modeler, by pressing the [Preview Simulation Set] button. The OPNET 'mod_dirs' preference will be used too for looking the file.

4. Values for promoted values (.ef): The number of files specified with the -ef option will be used for. building a list of supplementary environment file names (.ef) as follows. The file names should contains the model name followed by the suffix '_params.ef'. If the scenario model specified is 'mymodel': 'mymodel_params-01.ef', 'mymodel_params-02.ef', ... until the <num-file-params> specified value. If the value of <num-file-params> is 1 (unique file), the name will not include the index: 'mymodel_params.ef' All of these files should exist, and they will be used to define a simulation set, together with the different scenario models and seeds.

5. Trajectories for mobile nodes (.trj): All the .trj files assigned to the 'trajectory' property of mobile nodes in the network. This assignment can be done both directly in the model or in the .ef files for promoted attributes.

3.3.2 Output files

1. Output Vector files (.ov), generated by 'op_runsim': They contains vector data recorded from each simulation run. If the scenario model specified is (e.g.) 'mymodel', the generated files will be numerated with an numeric index: 'mymodel-001.ov', 'mymodel-002.ov', ...

2. Output Scalar files (.os), generated by 'op_runsim': They contains scalar statistic values recorded from one simulation, (min,max,...). It will be generated one .os file for each simulation run. If the scenario model specified is (e.g.) 'mymodel', the generated files will be numerated with an numeric index: 'mymodel-001.os', 'mymodel-002.os', ...

3. Output environment files (.ef), generated by 'op_runsim': It will be an environment file that will save attribute requests for each, simulation in the simulation set, that is, the real value assigned to each promoted attribute in the scenario model. It is automatically specified by 'simul_run' when invoking to 'op_runsim' with the '-attr_reqs_attr' option. If the scenario model specified is 'mymodel', the generated files will be: 'mymodel_promoted-001.ef', 'mymodel_promoted-002.ef', ... These files allow to check if the real values assigned to each of the promoted attributes match with the intended values especified in the input environment files (.ef).

4. Output log files (.log), generated by 'op_runsim': If the scenario model specified is 'mymodel', the generated files will be: 'mymodel-001.log', 'mymodel-002.log', ...

5. Animation history files (.ah), generated by 'op_runsim': When the option -anim_hist is specified, a file is generated during the the simulation containing the positions of each node at any time. If the scenario model specified is 'mymodel', the generated files will be: 'mymodel-001.ah', 'mymodel-002.ah', ... These animation history files (.ah) contain binary data, but they can be converted into animation script files (.as) encoded in an ASCII text file by using the 'op_cvanim -reverse' utility. In this way, they can be viewed within a standard text editor.

6. Output log file (.txt), generated by 'simul_run': This 'simul_run' program will generate a text file for each simulation run including some useful information, namely, the scenario model, seed, input/output files, complete command-line, etc. If the scenario model specified is 'mymodel', the generated files will be: 'mymodel-001.txt', 'mymodel-002.txt', ...

Output files using a cluster (Condor)

The *simul_run* program invokes to 'op_runsim' to run each simulation and waits until the end of that command to invoke it again, that is, simulations are executed sequentially. However, if a cluster Condor is used (`-cluster` option), the *simul_run* program generates automatically all the necessary files for launching a job for each simulation, and passes the control to Condor, finishing nearly immediately.

The `-cluster` option allows to choose between 3 possibilities for launching jobs in Condor. All of them have the same goal, but do it in a different way, generating more or less files: (1) Independent jobs, is the one that generates the greatest number of files, (2) DAG file, and (3) Parametrized DAG file, is the method that generates the fewest number of files. In the following each of the methods is detailed, showing an example of every file generated in each case.

Method 1 - Independent jobs In this method, both a .job file and a script file .sh are created for each of the simulations to run. The .job file contains all the necessary parameters for the simulation (executable, arguments, files to transfer to and from Condor, etc.), whereas the script .sh include only the command line to launch the previous .job file. Then, the *simul_run* program will invoke repeatly each script .sh for queueing each job. After a job is launched, Condor write 3 more files: (1) output messages (.out) from the executed program (op_runsim), (2) error messages and warnings (.err) from the executed program (op_runsim), and (3) messages from Condor (.log).

Thus, in addition to the files generated by the programs *simul_run* and 'op_runsim' (enumerated in the previous section), this method could generate a great number of files: two per simulation run (.sh, .job), generated by *simul_run*, plus three more files generated by Condor (.out, .err, .log) during the execution of each job.

For example, if the scenario is 'mymodel', these 5 extra files when using Condor will be named as following. Note that, for organization reasons, these files will be written to a subdirectory named 'condor1' under the output directory or working path (the number '1' at the end mean the number of method used).

1. script file: /condor1/mymodel-001.sh, ...
2. job file: /condor1/mymodel-001.job, ...
3. messages to stdout: /condor1/mymodel-001.out, ...
4. messages to stderr: /condor1/mymodel-001.err, ...
5. mesages from Condor: /condor1/mymodel-001.log, ...

As an example, for a specific simulation (e.g. #007), these 5 files would contain the following:

```
# -----------------------------------------------------
#  File:
#  ./condor1/mymodel-007.sh
# -----------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# -----------------------------------------------------
condor_submit ./condor1/mymodel-007.job
# -----------------------------------------------------


# -----------------------------------------------------
#  File:
#  ./condor1/mymodel-007.job
# -----------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# -----------------------------------------------------

Universe = vanilla

# Nombre del ejecutable y argumentos ...
Executable = $(JOB_EXE)
Arguments = $(JOB_ARGS)

# Variables de entorno utilizadas por el proceso
getenv =True
#environment = "PATH=/common/apps/opnet/16.0.A/sys/unix/bin"
environment  = "PATH=/common/apps/opnet/16.0.A/sys/pc_intel_linux/bin"

# Nombres para los ficheros de salida, error y log ...
Output = ./condor1/mymodel-007.out
Error  = ./condor1/mymodel-007.err
Log    = ./condor1/mymodel-007.log
```

```
# Configuracion para la transferencia de ficheros ...
Transfer_Files = Always
#should_transfer_files = Yes
#when_to_transfer_output = On_Exit

# Ficheros de entrada a transferir al nodo ...
Transfer-Input-Files = /home/user/op_admin/env_db16.0, mymodel.nt.m, mymodel.pb.m, myn
# Ficheros de salida a transferir desde el nodo ...
Transfer-Output-Files = mymodel_promoted-007.ef, ./results/mymodel-007.ov, ./results/n

# Notificaciones al usuario por correo electronico ...
notification = Never
notify_user = user@umh.es

# Ponemos en cola el proceso ...
Queue

# -----------------------------------------------------------------------------




<<< Warning >>>
  * Time:      13:41:52 Sun Jun 22 2008
  * Product:   Generic Product
  * Package:   Vos (Virtual Operating System)
  * Function:  Function Name Unavailable
  * Error:     Unable to set locale
               Spanish_Spain.1252
               File will be parsed using current system locale setting.

000 (68071.000.000) 05/13 19:26:46 Job submitted from host: <193.147.148.245:32775>
...
001 (68071.000.000) 05/13 19:26:49 Job executing on host: <192.168.100.23:33587>
...
005 (68071.000.000) 05/13 19:26:49 Job terminated.
(1) Normal termination (return value 1)
Usr 0 00:00:00, Sys 0 00:00:00  -  Run Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Total Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
84  -  Run Bytes Sent By Job
1593  -  Run Bytes Received By Job
84  -  Total Bytes Sent By Job
1593  -  Total Bytes Received By Job
...
```

Method 2 - DAG file Similarly to method 1, a .job file will be created for each simulation, but instead of creating a distinct script.sh for each one, in this case it will be created an unique script .sh, which will launch a unique .dag file. DAG (Directed Aciclic Graph) files allow to define a list of jobs files and a set of dependences parent-child between them, in such way that the children jobs will wait until their parent job finishes. In addition, DAG files have the advantage that allow to limit the maximum number of simultaneous jobs, which is necessary when there are more simulations to run that available OPNET Modeler licenses.

So, apart from the files generated by Condor (.out, .err, .log), this method reduce nearly to the half the number of generated files, because the number of .job files will be the same and only one .sh file will be generated.

For example, if the model is 'mymodel', files will be named as /condor2/mymodel.sh, /condor2/mymodel.dag, and the .job files will be identical as in method 1: /condor2/mymodel-001.job, ... About the files generated by Condor (.out, .err, .log), they will be similar to those in method 1, but named with a different subdirectory: /condor2/mymodel-001.out, ...

The contents of the unique script .sh and .dag files will be as shown below:

```
# ------------------------------------------------------
#  File:
#  ./condor2/mymodel.sh
# ------------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# ------------------------------------------------------
condor_submit_dag -f ./condor2/mymodel.dag -maxjobs 20
# ------------------------------------------------------


# ------------------------------------------------------
#  File:
#  ./condor2/mymodel.dag
# ------------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# ------------------------------------------------------

# ------------------------------------------------------
# 10 jobs
# ------------------------------------------------------

JOB P1 ./condor2/mymodel-001.job
JOB P1 ./condor2/mymodel-002.job
JOB P1 ./condor2/mymodel-003.job
...
JOB P1 ./condor2/mymodel-010.job

PARENT P1 CHILD P2,P3,P4,P5,P6,P7,P8,P9,P10

# ------------------------------------------------------
```

Method 3 - Parametrized DAG file In this last method, only 3 files will be generated, apart from the files generated by Condor (.out, .err, .log): a template .job which receives arguments from a .dag file, which contains the information of all the jobs to be launched, and finally, the script .sh file that executes the .dag file.

Similarly to the previous method 2, the maximun number of jobs can be limited if the number of simulations exceed the number of available OPNET Modeler licences.

If the model is 'mymodel', the three files will be named as, /condor3/mymodel.sh, /condor3/mymodel.dag, and /condor3/mymodel.job, and will contains:

```
# ------------------------------------------------------
#  File:
#  ./condor3/mymodel.sh
# ------------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# ------------------------------------------------------
condor_submit_dag -f ./condor3/mymodel.dag -maxjobs 20
# ------------------------------------------------------


# ------------------------------------------------------
#  File:
#  ./condor3/mymodel.dag
# ------------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# ------------------------------------------------------

# ------------------------------------------------------
# 10 jobs
# ------------------------------------------------------

JOB P1 ./condor3/mymodel.job
VARS P1 JOB_ID="001"
VARS P1 JOB_FILENAMES="mymodel"
VARS P1 JOB_EXE="/common/apps/opnet/16.0.A/sys/pc_intel_linux/bin/op_runsim"
VARS P1 JOB_ARGS="-net_name mymodel -seed 1 -duration 360 -ov_file ./results/mymodel-0
#VARS P1 JOB_ENV="PATH=/common/apps/opnet/16.0.A/sys/unix/bin"
VARS P1 JOB_ENV="PATH=/common/apps/opnet/16.0.A/sys/pc_intel_linux/bin"
VARS P1 JOB_USER_PARAM_1=""
VARS P1 JOB_USER_PARAM_2=""

...

JOB P10 ./condor3/mymodel.job
VARS P10 JOB_ID="010"
VARS P10 JOB_FILENAMES="mymodel"
VARS P10 JOB_EXE="/common/apps/opnet/16.0.A/sys/pc_intel_linux/bin/op_runsim"
VARS P1 JOB_ARGS="-net_name mymodel -seed 1 -duration 360 -ov_file ./results/mymodel-0
#VARS P10 JOB_ENV="PATH=/common/apps/opnet/16.0.A/sys/unix/bin"
VARS P10 JOB_ENV="PATH=/common/apps/opnet/16.0.A/sys/pc_intel_linux/bin"
VARS P10 JOB_USER_PARAM_1=""
VARS P10 JOB_USER_PARAM_2=""

PARENT P1 CHILD P2,P3,P4,P5,P6,P7,P8,P9,P10

# ------------------------------------------------------


# ------------------------------------------------------
#  File:
```

```
#  ./condor3/mymodel.job
# ------------------------------------------------------
#                ***  WARNING  ***
#  This file is automatically generated.
#  If you edit it, your edits will be discarded
#  next time the file is generated.
# ------------------------------------------------------

Universe = vanilla

# Nombre del ejecutable y argumentos ...
Executable = $(JOB_EXE)
Arguments = $(JOB_ARGS)

# Variables de entorno utilizadas por el proceso
getenv =True
environment = $(JOB_ENV)

# Nombres para los ficheros de salida, error y log ...
Output = ./condor3/$(JOB_FILENAMES)-$(JOB_ID).out
Error  = ./condor3/$(JOB_FILENAMES)-$(JOB_ID).err
Log    = ./condor3/$(JOB_FILENAMES)-$(JOB_ID).log

# Configuracion para la transferencia de ficheros ...
Transfer_Files = Always
#should_transfer_files = Yes
#when_to_transfer_output = On_Exit

# Ficheros de entrada a transferir al nodo ...
Transfer-Input-Files = /home/user/op_admin/env_db16.0, $(JOB_FILENAMES)$(JOB_USER_PARAM_
# Ficheros de salida a transferir desde el nodo ...
Transfer-Output-Files = $(JOB_FILENAMES)_promoted-$(JOB_ID).ef, ./results/$(JOB_FILENAME

# Notificaciones al usuario por correo electronico ...
notification = Never
notify_user = user@umh.es

# Ponemos en cola el proceso ...
Queue
# ------------------------------------------------------------------------------
```

3.4 Examples

The following examples show how to use the *simul_run* program to run some typical sequence of simulations. In the first one, we assume that the scenario is composed of a set of nodes, in which there are not any promoted attribute. We are going to run the simulation of the scenario as-is, repeating it with five different seeds (10, 20, 30, 40, 50) in order to achieve a greater confidence degree. The input file (scenario) is 'mymodel.nt.m', and we will need to specify the list with the option `-seeds` as shown:

```
simul_run -m mymodel -seeds 10,20,30,40,50
```

Please, note that the .nt.m suffix should not be specified at the end, because we are not specifying the file name but the corresponding model name. By default, the final .nt.m file used is determined by searching throught the paths listed in the `mod_dirs` preference within OPNET Modeler, exactly in that order. However, you can force the file to be used by using the `-i` opcion, in order to specify the unique directory where to look for the .nt.m file, as shown in the following example, without needing to modify it within the OPNET Modeler GUI. This directory can be either an obsolute or relative path to the working directory.

```
simul_run -m mymodel -seeds 10,20,30,40,50 -i ./paper2011/models
```

Similarly, it can be specified the output directory with the options `-o` or `-w`. That directory applies to the following output files: ... In the following example, such output files will be written to the subdirectory ./paper2011/results.

```
simul_run -m mymodel -seeds 10,20,30,40,50 -i ./paper2011/models
                                           -o ./paper2011/results
```

Two options used frecuently are `-duration` and `-probe_start_time`, which indicates the amount of time simulated and the start time for collecting the selected statistics, respectively. The `-probe_start_time` is intended for allowing the stabilization of the system. Both values are measured in seconds:

```
simul_run -m mymodel -duration 360 -probe_start_time 60
```

The *simul_run* program can run nearly silently, or showing more messages by specifying the `-v` option. On the other hand, by activating the option `-verbose_sim`, the `op_runsim` program will show periodic messages onto the screen during the simulations. It is equivalent to the similar option that `op_runsim` can receive, although in this case it is not necessary to add 'true' or 'false': if that option is specified to *simul_run*, it will be passed as `-verbose_sim true` to `op_runsim`; otherwise, it will be passed as `-verbose_sim false`. In case of using a cluster Condor, all of those messages will be written to the .out files generated by Condor.

In order to define sequences of simulations, in addition to the option `-seeds` explained above, two more options are available: `-modelsuffixes` (for using distint netwrok models named with a numeric suffix) and `-ef` (for setting different values to any promoted attribute). The sequence will have a number of simulations defined for the number of distinct combinations of those three lists, sorted by the scenario name, then the parameters file, and finally, the seed. The following example shows how to define a simulation set using three distinct scenarios, 5 environment files (.ef), and 5 seeds. So, the sequence will be composed of 75 simulations (3x5x5).

```
simul_run -m mymodel -modelsuffixes 1..3  -ef 5
                     -seeds 10,20,30,40,50   -c
```

About the values specified with the options `-modelsuffixes` and `-seeds`, they can be specified both by using an enumeration of values or a range between two values, as can be seen in the previous example. Note that it has been appended the option `-c` at the end, which force all the necessary models and object files to

be recompiled. That option is recommended prior executing a simulation set in order to be sure that all the files are update. That option is only passed through to `op_runsim` in the first simulation. Another posibility would be to manually execute the command `op_mko -all` before launching the sequence.

It is necessary that the following files exist either in some of the paths defined in the 'mod_dirs' preference of OPNET Modeler, or in the path specified with the option `-i`:

1. Scenario models (x3): mymodel-01.nt.m, mymodel-02.nt.m, mymodel-03.nt.m
2. Stadistics (x3): mymodel-01.pb.m, mymodel-02.pb.m, mymodel-03.pb.m
3. Environment file for DES Parameters (x1): mymodel_des.ef
4. Environment files for promoted values (x5): mymodel_params-01.ef, mymodel_params-02.ef, mymodel_params-03.ef, mymodel_params-04.ef, mymodel_params-05.ef

When executing a sequence of simulations it is possible that some of the simulations fail, or the user cancel it for any reason. In those cases, it could be useful to run a specific simulation, or starting from the last simulation failed or aborted, instead of repeating all the simulation set. This can be achieved by specifying an interval with the options `-first` and `-last`. For example, the previous example run a simulation set composed of 75 simulations; if for any reason we need to repeat the five ones (71, 72, 73, 74, and 75), we can append the parameter `-first 71` to the command line.

If the time required to run a simulation set is high, it is recommended to run the simulations in parallel if a cluster Condor is available with multiple processors. The program *simul_run* allows to make use of the cluster easily only by appending the option `-cluster` with the appropiate arguments. So, the user do not need the knowledge needed to launch jobs in such cluster. The arguments for this option are the number of processors (i.e. the maximum number of simulations simultaneously run), and the method used to launch jobs (explained previously in the section 3.3.2). With regards of the number of processors, it is important to note that there exists the limitation of the number of OPNET Modeler licences available, as each simulation need one. In other words, if our cluster has 96 processors and we own 20 OPNET Modeler licences available, the maximum value specificed should be 20; a greater value will make that many simulations fail because of they can not get a license to run. The `-cluster` option admits two additional arguments in order to notify to the e-mail specified when some event occur, for example, when a process finalizes succesfully or aborted. By default, none event is notified, as in the following example.

```
simul_run -m mymodel -modelsuffixes 1..3 -ef 5
                     -seeds 10,20,30,40,50  -c  -cluster 20 3
```

Finally, *simul_run* offers the posibility to specify the support for 64-bit platforms (by default 32-bit) with two mutually exclusive options: `-m32` and `-m64`. The distributed execution using several processors is also allowed by specifying the `-distributed` option, thanks to corresponfing options that OPNET Modeler offers for that purpose (still under development).

Bibliography

[1] Inc. OPNET Technologies. Opnet modeler.

[2] The Condor Team. Condor.